1 Purpose

nag_ztrrfs (f07tvc) returns error bounds for the solution of a complex triangular system of linear equations with multiple right-hand sides, $AX = B$, $A^TX = B$ or $A^HX = B$.

2 Specification

```c
#include <nag.h>
#include <nagf07.h>

void nag_ztrrfs (Nag_OrderType order, Nag_UploType uplo,
                 Nag_TransType trans, Nag_DiagType diag, Integer n, Integer nrhs,
                 const Complex a[], Integer pda, const Complex b[], Integer pdb,
                 const Complex x[], Integer pdx, double ferr[], double berr[],
                 NagError *fail)
```

3 Description

nag_ztrrfs (f07tvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex triangular system of linear equations with multiple right-hand sides $AX = B$, $A^TX = B$ or $A^HX = B$. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_ztrrfs (f07tvc) in terms of a single right-hand side $b$ and solution $x$.

Given a computed solution $x$, the function computes the component-wise backward error $\beta$. This is the size of the smallest relative perturbation in each element of $A$ and $b$ such that $x$ is the exact solution of a perturbed system

$$(A + \delta A)x = b + \delta b$$

$$|\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.$$  

Then the function estimates a bound for the component-wise forward error in the computed solution, defined by:

$$\max_i |x_i - \hat{x}_i| / \max_i |x_i|$$

where $\hat{x}$ is the true solution.

For details of the method, see the f07 Chapter Introduction.

4 References


5 Arguments

1: order – Nag_OrderType

   Input

   On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.
2:  **uplo** – Nag_UploType  
*Input*  
*On entry:* specifies whether $A$ is upper or lower triangular.  

- **uplo** = Nag_Upper  
  $A$ is upper triangular.  
- **uplo** = Nag_Lower  
  $A$ is lower triangular.  

*Constraint:* **uplo** = Nag_Upper or Nag_Lower.

3:  **trans** – Nag_TransType  
*Input*  
*On entry:* indicates the form of the equations.  

- **trans** = Nag_NoTrans  
  The equations are of the form $AX = B$.  
- **trans** = Nag_Trans  
  The equations are of the form $A^T X = B$.  
- **trans** = Nag_ConjTrans  
  The equations are of the form $A^H X = B$.  

*Constraint:* **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4:  **diag** – Nag_DiagType  
*Input*  
*On entry:* indicates whether $A$ is a nonunit or unit triangular matrix.  

- **diag** = Nag_NonUnitDiag  
  $A$ is a nonunit triangular matrix.  
- **diag** = Nag_UnitDiag  
  $A$ is a unit triangular matrix; the diagonal elements are not referenced and are assumed to be 1.  

*Constraint:* **diag** = Nag_NonUnitDiag or Nag_UnitDiag.

5:  **n** – Integer  
*Input*  
*On entry:* $n$, the order of the matrix $A$.  

*Constraint:* $n \geq 0$.

6:  **nrhs** – Integer  
*Input*  
*On entry:* $r$, the number of right-hand sides.  

*Constraint:* **nrhs** $\geq 0$.

7:  **a[dim]** – const Complex  
*Input*  
*Note:* the dimension, $dim$, of the array $a$ must be at least $\max(1, pda \times n)$.  

*On entry:* the $n$ by $n$ triangular matrix $A$.  
If **order** = Nag_ColMajor, $A_{ij}$ is stored in $a[(j - 1) \times pda + i - 1]$.  
If **order** = Nag_RowMajor, $A_{ij}$ is stored in $a[(i - 1) \times pda + j - 1]$.  
If **uplo** = Nag_Upper, the upper triangular part of $A$ must be stored and the elements of the array below the diagonal are not referenced.  
If **uplo** = Nag_Lower, the lower triangular part of $A$ must be stored and the elements of the array above the diagonal are not referenced.  
If **diag** = Nag_UnitDiag, the diagonal elements of $A$ are assumed to be 1, and are not referenced.
8:    pda – Integer

   Input

   On entry: the stride separating row or column elements (depending on the value of order) of the matrix A in the array a.

   Constraint: pda ≥ max(1, n).

9:    b[dim] – const Complex

   Input

   Note: the dimension, dim, of the array b must be at least

   max(1, pdb × nrhs) when order = Nag_ColMajor;
   max(1, n × pdb) when order = Nag_RowMajor.

   The (i, j)th element of the matrix B is stored in

   b[(i - 1) × pdb + j - 1] when order = Nag_ColMajor;
   b[(j - 1) × pdb + i - 1] when order = Nag_RowMajor.

   On entry: the n by r right-hand side matrix B.

10:   pdb – Integer

       Input

       On entry: the stride separating row or column elements (depending on the value of order) in the array b.

       Constraints:

       if order = Nag_ColMajor, pdb ≥ max(1, n);
       if order = Nag_RowMajor, pdb ≥ max(1, nrhs).

11:   x[dim] – const Complex

       Input

       Note: the dimension, dim, of the array x must be at least

       max(1, pdx × nrhs) when order = Nag_ColMajor;
       max(1, n × pdx) when order = Nag_RowMajor.

       The (i, j)th element of the matrix X is stored in

       x[(i - 1) × pdx + j - 1] when order = Nag_ColMajor;
       x[(j - 1) × pdx + i - 1] when order = Nag_RowMajor.

       On entry: the n by r solution matrix X, as returned by nag_ztrtrs (f07tsc).

12:   pdx – Integer

       Input

       On entry: the stride separating row or column elements (depending on the value of order) in the array x.

       Constraints:

       if order = Nag_ColMajor, pdx ≥ max(1, n);
       if order = Nag_RowMajor, pdx ≥ max(1, nrhs).

13:   ferr[nrhs] – double

       Output

       On exit: ferr[j - 1] contains an estimated error bound for the jth solution vector, that is, the jth column of X, for j = 1, 2, ..., r.

14:   berr[nrhs] – double

       Output

       On exit: berr[j - 1] contains the component-wise backward error bound β for the jth solution vector, that is, the jth column of X, for j = 1, 2, ..., r.

15:   fail – NagError*

       Input/Output

       The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument \(\langle\text{value}\rangle\) had an illegal value.

NE_INT

On entry, \(n = \langle\text{value}\rangle\).
Constraint: \(n \geq 0\).

On entry, \(\text{nrhs} = \langle\text{value}\rangle\).
Constraint: \(\text{nrhs} \geq 0\).

On entry, \(\text{pda} = \langle\text{value}\rangle\).
Constraint: \(\text{pda} > 0\).

On entry, \(\text{pdb} = \langle\text{value}\rangle\).
Constraint: \(\text{pdb} > 0\).

On entry, \(\text{pdx} = \langle\text{value}\rangle\).
Constraint: \(\text{pdx} > 0\).

NE_INT_2

On entry, \(\text{pda} = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(\text{pda} \geq \max(1, n)\).

On entry, \(\text{pdb} = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(\text{pdb} \geq \max(1, n)\).

On entry, \(\text{pdb} = \langle\text{value}\rangle\) and \(\text{nrhs} = \langle\text{value}\rangle\).
Constraint: \(\text{pdb} \geq \max(1, \text{nrhs})\).

On entry, \(\text{pdx} = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(\text{pdx} \geq \max(1, n)\).

On entry, \(\text{pdx} = \langle\text{value}\rangle\) and \(\text{nrhs} = \langle\text{value}\rangle\).
Constraint: \(\text{pdx} \geq \max(1, \text{nrhs})\).

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

The bounds returned in \texttt{ferr} are not rigorous, because they are estimated, not computed exactly; but in
practice they almost always overestimate the actual error.
8 Parallelism and Performance

nag_ztrrfs (f07tvc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ztrrfs (f07tvc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_ztrrfs (f07tvc), for each right-hand side, involves solving a number of systems of linear equations of the form $Ax = b$ or $A^Hx = b$; the number is usually 5 and never more than 11. Each solution involves approximately $4n^2$ real floating-point operations.

The real analogue of this function is nag_dtrrfs (f07thc).

10 Example

This example solves the system of equations $AX = B$ and to compute forward and backward error bounds, where

$$A = \begin{pmatrix}
4.78 + 4.56i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.00 - 0.30i & -4.11 + 1.25i & 0.00 + 0.00i & 0.00 + 0.00i \\
2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & 0.00 + 0.00i \\
-1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i
\end{pmatrix}$$

and

$$B = \begin{pmatrix}
-14.78 - 32.36i & -18.02 + 28.46i \\
2.98 - 2.14i & 14.22 + 15.42i \\
-20.96 + 17.06i & 5.62 + 35.89i \\
9.54 + 9.91i & -16.46 - 1.73i
\end{pmatrix}.$$
#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda +I-1]
#define B(I, J) b[(J-1)*pdb +I-1]
#define X(I, J) x[(J-1)*pdx +I-1]
#else
#define A(I, J) a[(I-1)*pda +J-1]
#define B(I, J) b[(I-1)*pdb +J-1]
#define X(I, J) x[(I-1)*pdx +J-1]
#endif
order = Nag_ColMajor;
#else
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
printf("nag_ztrrfs (f07tvc) Example Program Results\n");
/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n ]");
#else
scanf("%*[\n ]");
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n ]", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n ]", &n, &nrhs);
#endif
berr_len = nrhs;
ferr_len = nrhs;
#ifdef NAG_COLUMN_MAJOR
pda = n;
pdb = n;
pdx = n;
#else
pda = n;
pdb = nrhs;
pdx = nrhs;
#endif
/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, Complex)) ||
    !(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * nrhs, Complex)) ||
    !(berr = NAG_ALLOC(berr_len, double)) ||
    !(ferr = NAG_ALLOC(ferr_len, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
/* Read A and B from data file, and copy B to X */
#ifdef _WIN32
scanf_s(" %39s%*[\n ]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[\n ]", nag_enum_arg);
#endif
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        {
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
        }
    }
}
```c
#ifdef _WIN32
    scanf_s("%*[\n"]
#else
    scanf("%*[\n"]
#endif
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
        #else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
        #endif
    }
#endif
    scanf("%*[\n"]
#else
    scanf("%*[\n"]
#endif
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
    #ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    #else
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    #endif
    }
#endif
    scanf("%*[\n"]
#else
    scanf("%*[\n"]
#endif
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
    {
        X(i, j).re = B(i, j).re;
        X(i, j).im = B(i, j).im;
    }
}
/* Compute solution in the array X */
/* nag_ztrtrs (f07tsc). */
/* Solution of complex triangular system of linear */
/* equations, multiple right-hand sides */
/* nag_ztrtrs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n, */
/* nrhs, a, pda, x, pdx, &fail); */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
/* Compute backward errors and estimated bounds on the */
/* forward errors */
/* nag_ztrrfs (f07tvc). */
/* Error bounds for solution of complex triangular system of */
/* linear equations, multiple right-hand sides */
/* nag_ztrrfs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n, */
/* nrhs, a, pda, b, pdb, x, pdx, ferr, berr, &fail); */
if (fail.code != NE_NOERROR)
```
Error from nag_ztrrfs (f07tvc).

Print solution

Print complex general matrix (comprehensive)

Print solution

Print complex general matrix (comprehensive)

Backward errors (machine-dependent)

Estimated forward error bounds

Program Data

nag_ztrrfs (f07tvc) Example Program Data

10.2 Program Data
10.3 Program Results

nag_ztrrfs (f07tvc) Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-5.0000, -2.0000)</td>
<td>(1.0000, 5.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(-3.0000, -1.0000)</td>
<td>(-2.0000, -2.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(2.0000, 1.0000)</td>
<td>(3.0000, 4.0000)</td>
</tr>
<tr>
<td>4</td>
<td>(4.0000, 3.0000)</td>
<td>(4.0000, -3.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

- 6.2e-17
- 2.7e-17

Estimated forward error bounds (machine-dependent)

- 2.9e-14
- 3.2e-14